J Log Lang Inf (2008) 17:43–68
DOI 10.1007/s10849-006-9031-0

ORIGINAL ARTICLE

First-Order Glue

Miltiadis Kokkonidis

Received: 26 July 2006 / Accepted: 23 November 2006 / Published online: 29 June 2007

Abstract Glue has evolved significantly during the past decade. Although the recent move to type-theoretic notation was a step in the right direction, basing the current Glue system on System F (second-order λ-calculus) was an unfortunate choice. An extension to two sorts and ad hoc restrictions were necessary to avoid inappropriate composition of meanings. As a result, the current system is unnecessarily complicated. A first-order Glue system is hereby proposed as its replacement. This new system is not only simpler and more elegant, as it captures the exact requirements for Glue-style compositionality without ad hoc improvisations, but it also turns out to be more powerful than the current two-sorted (pseudo-) second-order system. First-order Glue supports all existing Glue analyses as well as more elegant alternatives. It also supports new, more demanding analyses.

Keywords Glue · Compositional semantics · Syntax-semantics interface · First-Order Linear Logic · Typing systems

1 Introduction

The principle of compositionality (Janssen, 1997) has played an important role in practically all post-Montague work on formal natural language semantics. Many researchers share Montague's belief that compositionality is an essential semantic principle, but feel that his version is rather limiting; in recent research, Montague's version of compositionality (Montague, 1974) has typically been replaced by more relaxed alternatives.

Dalrymple, Lamping, and Saraswat (1993) argued for a compositional semantics framework in LFG (Dalrymple, 2001; Kaplan & Bresnan, 1982) based primarily on functional structure, a semantic projection function and linear logic as a glue language

M. Kokkonidis (✉)
Computational Linguistics Group, University of Oxford, Walton Street, Oxford, OX1 2HG, UK
e-mail: miltiadis.kokkonidis@clg.ox.ac.uk

for putting together the meaning of a sentence or a phrase from the meanings of its parts. The motivation for this line of research is that having a resource logic driving semantic composition fits well with the unordered nature of f-structure while preserving the fundamental semantic principle (underlying also Montague-style compositionality) that the meaning of a phrase is formed by a combination of the meanings of each and every meaning component used exactly once. Being resource sensitive, Glue enforces a semantic version of the completeness and coherence principles (Dalrymple, 2001).

Glue has the ability to derive the various readings of a sentence given its semantic atoms. It is essentially a theorem prover. Various 'transformations' such as currying/uncurrying and type-lifting are proof steps performed wherever needed. It follows that uniform worst-case typing is unnecessary in Glue. For example, Glue allows proper names to be treated as simple entities rather than as second-order predicates; type uniformity of proper names with quantified noun phrases is not necessary. In many ways Glue derivations are similar to the type-driven derivations in the Lambek–van Benthem tradition of Categorial Grammar, or Type-Logical Grammar as we will be referring to it (Morril, 1994; Moorgat, 1997). CG deals with issues of syntax and semantics together; this is both the source of its elegance and a potential problem that has always been the subject of CG research. Glue only deals with semantic assembly, leaving syntactic issues to other parts of the framework within which it is used. Although, originally designed for LFG, it has also been successfully integrated with LTAG (Frank & van Genabith, 2001), HPSG (Asudeh & Crouch, 2002), CG (Asudeh & Crouch, 2001) and CFG (Asudeh & Crouch, 2001).

The first and foremost challenge for a derivational approach is to get all possible readings of a sentence while avoiding erroneous interpretations. This requires making the right choices on two levels: that of the formal system and that of the analyses of linguistic data within it. Sometimes the desire for new analyses that address shortcomings of earlier approaches brings about changes to the formal system, usually making it more complicated and less elegant. CG is a popular example.

A remarkable fact about Glue is that both the change to type-theoretic notation (Dalrymple, Gupta, Pereira, & Saraswat, 1997a) and the change to a first-order system proposed here were not motivated by a need for more power and new logical operators or other formal machinery that would help with certain analyses. The foundational intuitions behind Glue as laid out by Dalrymple, Lamping, and Saraswat (1993) and especially in subsequent work (Dalrymple, Lamping, Pereira, & Saraswat, 1995a,b, 1997b) have successfully captured the requirements of a programme of compositional semantics for LFG (and other grammar formalisms). As a result, they have not been the subject of revision and remain the basis for motivating, defining and understanding Glue. Indeed, one of the claims in this paper is that the proposed system captures these intuitions in the most direct way possible. As a result, it is not more complicated than the one it seeks to replace. On the contrary, it is both simpler and more powerful.

While the foundational intuitions behind Glue have changed very little since its first appearance and have remained intact since early subsequent work, the formal system for Glue has been evolving. The original Glue system (G0) of Dalrymple, Lamping, and Saraswat (1993) was quickly superseded by the system later introduced by Dalrymple et al. (1995a). The system presented there (G1) was in many ways a simplification of G0 as it did away with rules and the use of the linear logic '!' modality. G1 was the system that brought Glue into the mainstream and much research has been carried out using it. The latest development was a new notation for Glue based on Sys-

tem F: Dalrymple et al. (1997a) aimed to explore the relation between Glue and CG approaches to the syntax-semantics interface and as a result introduced G2, a Glue system closer to systems used in the TLG camp.

Although G2 was presented as a system that only served as a bridge between the LFG/Glue world and the TLG world, it later became the prominent Glue system, replacing G1 as the system of choice among Glue researchers. Its type-theoretic notation was neater, more concise, and more readable than the notation of G1. An important feature was the separation of assembled meaning from the linear logic types that controlled the derivation. Moreover, G2 notation was already familiar to its intended audience as typed λ-calculi are commonplace in the formal semantics field. Familiarity with its notation also helped G2 make Glue more accessible to a much wider audience that included logicians and computer scientists. In the present paper, I argue for the next step in the evolution of Glue: a simple first-order system (G3) that improves on the current one (G2).

In Sect. 2, we will see how, given a minimal set of assumptions and methodological choices, the First-Order Glue approach emerges as a candidate for carrying out a programme of compositional[1] semantics. As the idea of a type system for meaning assembly is developed, we reach a cross-roads. One direction leads to G2 and the other to G3. G2 has been used extensively in the Glue research literature. The G3 direction had never before been considered.[2] In Sect. 3, we will see that, for all its System F (second-order typed λ-calculus) notation and claimed heritage, G2 is equivalent to a restricted version of the proposed first-order Glue system. Every G2 derivation can be encoded in a fragment of G3. Moreover, the computational properties of G2 are preserved for analyses transferred from G2 to the restricted fragment of G3. In Sect. 4, we will examine the common characteristics of the design of G2 and that of G3. We will also consider why the various ad hoc innovations of G2 are needed to obtain the behaviour that comes naturally in first-order Glue (G3). Discussions in earlier sections lead to the conclusions drawn in Sect. 5; there, I claim that G3 is a step forwards in the evolution of Glue and propose that this step be made.

2 Towards first-order Glue

The proposed Glue system is a term-assignment system for a fragment of first-order intuitionistic linear logic. Both the design decision to use linear-logic and the decision to use predicates as types follow from fairly basic observations and assumptions in the spirit of Montague's work. We will see how one can get from those assumptions and observations to the proposed Glue system, thus motivating its design.

Richard walks. (1)

Richard walks slowly. (2)

Regarding a sentence such as (1) as a truth statement, we give the semantic expression that corresponds to it an appropriate type for truth statements; we call this type *t*. We also make the assumption that there is a systematic process that produces the

[1] The term 'compositional' is not used in this paper for a programme of semantics faithful to the letter of Montague's work, but to a broader programme of which Montague's work is a particular flavour.

[2] First-Order Glue first appeared in unpublished work (M. Kokkonidis, 2003) from which the current work derives.

Table 1 Type assignments in a system based on intuitionistic propositional logic

Category	Type
Proper name	e
Determiner	$(e \to t) \to (e \to t) \to t$
Common noun	$e \to t$
Adjective	$(e \to t) \to (e \to t)$
S adverb	$t \to t$
VP adverb	$(e \to t) \to (e \to t)$
Intransitive verb	$e \to t$
Transitive verb	$e \times e \to t$

meaning(s) of a sentence from the semantic contributions of the words that appear in it and certain syntactic structures, such as the relative clause (Dalrymple, 2001), or other factors that will not concern us here, such as ironic tone in the speaker's voice.

So, we should ask what the semantic contributions of 'Richard' and 'walks' are and if there are any semantic contributions due to special syntactic constructs. A methodological preference in Glue and other similar systems is to put emphasis on the words of a sentence and only search for meaning contributions in syntactic constructs when absolutely necessary. In (1), there are two words. The first, 'Richard', denotes an entity and we may assign the type e to its semantic contribution. Then, assuming no other semantic contribution has been made by the sentence and that our system does not rely on rules that combine, for example, something of type e and something of, say, type *iv* (intransitive verb), to give something of type t, it is up to the semantic contribution of 'walks' to return something of type t given something of type e. Reasoning that way, we may decide to give the semantic contribution of 'walks' the type $e \to t$.

We can treat modification in a similar fashion. The difference between (1) and (2) is the adverbial modifier 'slowly'. Given that the semantic contributions of 'Richard' and 'walks' can be combined into something of type t in the way just described and that the meaning of (2) will also be of type t, the semantic contribution of 'slowly' has a type that explains why if it is omitted from (2) the resulting sentence (1) would be well-formed, whereas if 'Richard' or 'walks' were omitted the same would not be true. Some adverbs are sentence modifiers ($t \to t$). Some, including 'slowly' here, are verb phrase modifiers ($(e \to t) \to (e \to t)$).

Writing the semantic atoms in (1) and (2) and their types on the left of a turnstile ('$\vdash$') and the term that is obtained by putting them together and its type on the right, we have respectively:

$$R, W \vdash \mathit{walk}\ \ \mathit{richard} : t, \tag{3}$$

and

$$R, W, S \vdash \mathit{slowly}\ \ \mathit{walk}\ \ \mathit{richard} : t, \tag{4}$$

where

$$R = \mathit{richard} : e,\ W = \mathit{walk} : e \to t,\ S = \mathit{slowly} : (e \to t) \to (e \to t).$$

Function application was sufficient for composing the meaning of (1) and (2). But this is not always the case:

Fig. 1 Typing rules for the $(\times, \rightarrow)$ fragment of the simply-typed λ-calculus

$$\frac{\Gamma \vdash E : T'}{\Gamma,\ N : T \vdash E : T'}\ (Weakening)$$

$$\frac{\Gamma,\ N : T,\ N : T \vdash E : T'}{\Gamma,\ N : T \vdash E : T'}\ (Contraction)$$

$$\frac{N : T,\ \Gamma,\ N' : T',\ \Gamma' \vdash E : T''}{N' : T',\ \Gamma,\ N : T,\ \Gamma' \vdash E : T''}\ (Exchange)$$

$$N : T \vdash N : T\ (Axiom)$$

$$\frac{\Gamma, X : T \vdash E : T'}{\Gamma \vdash \lambda X.E : T \rightarrow T'}\ (\rightarrow Intro.)$$

$$\frac{\Gamma \vdash E : T' \rightarrow T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash E\ E' : T}\ (\rightarrow Elim.)$$

$$\frac{\Gamma \vdash E : T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash (E, E') : T \times T'}\ (\times Intro.)$$

$$\frac{\Gamma, X : T, X' : T' \vdash E : T'' \quad \Gamma' \vdash E' : T \times T'}{\Gamma, \Gamma' \vdash \mathbf{let}\ (X, X') = E'\ \mathbf{in}\ E : T''}\ (\times Elim.)$$

The cat likes Alonso. (5)

$$\begin{aligned} \mathit{the} &: (e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow t, \\ \mathit{cat} &: e \rightarrow t, \\ \mathit{like} &: e \times e \rightarrow t, \\ \mathit{alonso} &: e. \end{aligned} \tag{6}$$

The only readily available combination by function application of the semantic atoms (6) of sentence (5) is that of the atoms contributed by the two components of the subject noun phrase. The result is its composite meaning. This, in turn, needs a predicate $(e \rightarrow t)$ argument in order to return a complete sentence (t).

$$\begin{aligned} \mathit{the\ cat} &: (e \rightarrow t) \rightarrow t, \\ \mathit{like} &: e \times e \rightarrow t, \\ \mathit{alonso} &: e. \end{aligned} \tag{7}$$

Whereas the transitive verb *like* expects a subject–object pair of entities as its argument, only the object entity, *alonso*, is readily available. Given a subject entity x, it becomes possible to form the pair *(x, alonso)* and subsequently, using that pair as the argument of *like*, the term *like (x, alonso)*. What was just described is a function from an unknown subject entity x to the truth value *like (x, alonso)*. This is written, using familiar λ-calculus notation, as $\lambda x.$ *like (x, alonso)*. This function is an appropriate argument for the subject NP meaning (*the cat*); combining the two gives a complete sentence meaning:[3]

$$\mathit{the\ cat}\ \lambda x.\ \mathit{like}\ (x, \mathit{alonso}) : t. \tag{8}$$

[3] A λ-abstraction term extends as much as it can: $\lambda x.\, E_1\ E_2 \ldots E_n$ is the same as $\lambda x.\, (E_1\ E_2\ \ldots\ E_n)$, not $(\lambda x.\, E_1)\ E_2\ \ldots\ E_n$.

The reasoning underlying the process of combining the elemental semantic contributions of sentence (5) into its composite semantic representation (8) involved ontology (e and t types) as well as grammatical roles (subject and object). It corresponds rather well to the following type derivation in the $(\rightarrow, \times)$ fragment of the simply-typed λ-calculus (Fig. 1) except that the later makes no mention of grammatical roles:

$$\dfrac{\dfrac{T \vdash T \quad C \vdash C}{T, C \vdash \mathit{the\ cat} : (e \rightarrow t) \rightarrow t} \quad \dfrac{\dfrac{L \vdash L \quad \dfrac{x : e \vdash x : e \quad A \vdash A}{A, x : e \vdash (x, \mathit{alonso}) : e \times e}}{L, A, x : e \vdash \mathit{like}\ (x, \mathit{alonso}) : t}}{L, A \vdash \lambda x.\, \mathit{like}\ (x, \mathit{alonso}) : e \rightarrow t}}{T, C, L, A \vdash \mathit{the\ cat}\ \lambda x.\, \mathit{like}\ (x, \mathit{alonso}) : t} \tag{9}$$

where $T = \mathit{the} : (e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow t$,
$C = \mathit{cat} : e \rightarrow t$,
$L = \mathit{like} : e \times e \rightarrow t$,
$A = \mathit{alonso} : e$.

The observation that the type derivation essentially follows the same kind of reasoning we used for putting together the given elemental semantic contributions into larger meaning expressions, especially the part that allowed us to build an $e \rightarrow t$ function from the elemental semantic contributions of 'like' and 'Alonso', is very important. Type systems typically serve as mechanisms for checking the well-formedness of the compositionally derived semantic expressions for a sentence. Therefore, it is not that surprising that there appears to be a relation between the type derivation for the type of the meaning term of a sentence and the reasoning that led to the formation of that meaning term. But can a type system become the mechanism for meaning composition? TLG and Glue research give an affirmative answer. However, such a type system must be capable of enforcing certain constraints of meaning composition that are irrelevant to a type system of meaning expressions. For instance, expressing bound-variable anaphora in a meaning-representation language may involve a variable appearing multiple times. This is a matter of that formal language's type system. But whether a semantic contribution can be reused in forming a composite meaning is a matter of the meaning-composition type system. It is this kind of typing system that is of interest here.

Our method of composing the meaning of sentences will be this: after taking all meaning placeholders (such as *alonso*, *like*, etc.) with their assigned compositional (Glue) types as the typing context, we let the type system find all distinct (up to α-equivalence) $\beta\eta$-irreducible terms that have the type t in that context. Replacing meaning placeholders with their corresponding expressions in some meaning-representation language should give well-formed expressions in that language.[4] Well-typedness may have to be established by structural induction on derivations. In some cases it is obvious that it is guaranteed and no such proof is explicitly given.

Even if our Glue analyses guarantee well-formedness, problems can still arise. One is that the type system and/or the type assignments may be too rigid, thus not allowing all readings to be obtained. The other is that derivations may lead to expressions that

[4] Meaning components do not affect composition; only their types do. Using (atomic) meaning placeholders (not the common practice in Glue literature) emphasises that fact. It also shows more clearly what the derivation is, i.e. what it is that Glue does. This is convenient for a comparison of G2 and G3 devoid of unnecessary reference to any particular meaning-representation language.

are not readings of the given sentence. To avoid this second problem, the type-system must capture the linguistic constraints of meaning assembly. We will see that a type system based on intuitionistic prepositional logic does not, but one based on linear predicate logic does. Crucially, it manages to do so without being too rigid.

The first constraint that intuitionistic logic does not capture is that each semantic contribution is to be used exactly once in the forming of the composite sentence meaning. The Weakening rule found in intuitionistic logic and type systems based on it [5] means that semantic contributions may be (incorrectly) omitted. For example, the semantic contributions of sentence (10) can be used to derive a truth statement that captures the meaning of (11). Clearly, this is undesirable.

Mary sings happily. (10)

$$\dfrac{\dfrac{mary : e \vdash mary : e \quad sing : e \to t \vdash sing : e \to t}{mary : e, sing : e \to t \vdash sing\ mary : t}\ {}^{\to_E}}{{}^{*}mary : e, sing : e \to t, happily : (e \to t) \to (e \to t) \vdash sing\ mary : t}\ \textit{Weakening}$$

Mary sings. (11)

The Contraction rule also found in intuitionistic logic allows a semantic contribution by a part of a sentence[6] to be used any number of times in the composite semantic representation for the sentence. The following example shows why Contraction is problematic for our purposes:

Vijay complains that John sings loudly. (12)

$$\dfrac{\dfrac{\vdots}{\begin{array}{l} vijay : e,\ complain : e \times t \to t, \\ \quad john : e,\ sing : e \to t, \\ \quad loudly : (e \to t) \to (e \to t), \\ \quad loudly : (e \to t) \to (e \to t) \end{array} \vdash \begin{array}{l} loudly\ (\lambda x.complain(x, \\ \quad loudly\ sing\ john))\ vijay \end{array} : t}}{\begin{array}{l} vijay : e,\ complain : e \times t \to t, \\ {}^{*}\ john : e,\ sing : e \to t, \\ \quad loudly : (e \to t) \to (e \to t) \end{array} \vdash \begin{array}{l} loudly\ (\lambda x.complain(x, \\ \quad loudly\ sing\ john))\ vijay \end{array} : t}\ \textit{Contraction}$$

Vijay complains loudly that John sings loudly. (13)

Linear logic (Girard, 1987) rejects the Weakening and Contraction rules. This guarantees that each semantic contribution (assumption) is used exactly once in the composite semantic representation (conclusion). Special modalities (? and !) are introduced to recover the same effect only where that is desirable. Resource sensitivity in linear logic also comes with a split of the traditional conjunction and disjunction connectives. There are two conjunctions, $\otimes$ (times) and & (with), and two disjunctions, $\oplus$ (plus) and $\parr$ (par). There are also four constants, $\mathbf{1}, \bot, \top, \mathbf{0}$, which are, respectively, the

[5] There is a famous result known as the Curry–Howard isomorphism (Howard, 1980) linking logics to type systems. It started with the type system of the simply-typed λ-calculus and propositional intuitionistic logic, but a Curry–Howard isomorphism exists for various other logic-type system pairs.

[6] Contraction does not allow the use of arbitrary 'invisible' assumptions in the proof process—this would defeat the purpose of any logic. It only allows an *existing* assumption to be used an arbitrary number of times.

neutral elements of the aforementioned four conjunctive and disjunctive connectives. Negation ($^{\perp}$) can defined by De Morgan equations. Double linear negation, like its classical but unlike its intuitionistic counterpart, cancels itself out. Linear implication can also be defined in terms of negation and the par connective in the exact same way implication can be defined in terms of negation and disjunction in both intuitionistic and classical logic. Finally, the quantifiers $\forall$ and $\exists$ behave (from a sequent calculus viewpoint) similarly to their intuitionistic and classical counterparts.[7]

Linear logic is very expressive and powerful, but for our purposes we only need the $(\otimes, \multimap, \forall)$ fragment of linear predicate logic. The first-order universal quantifier of G3 corresponds to a certain flavour of type polymorphism that is more appropriate for our purposes than the polymorphism supported by the second-order quantification of System F, the system on which Dalrymple et al. (1997b) based G2. However, given our discussion so far, it is not obvious why we need predicate rather than propositional logic to be the basis of our type system.

As a matter of fact, a system based on the $(\otimes, \multimap)$ fragment of linear propositional logic suffices to avoid the problems Weakening and Contraction may cause and is the natural next step in our quest to find a type system that captures compositionality constraints now that we have rejected the $(\times, \rightarrow)$ fragment of propositional intuitionistic logic. All one has to do to move from the intuitionistic system to its linear counterpart is replace $\rightarrow$ with $\multimap$ and $\times$ with $\otimes$ and remember not to use Weakening and Contraction in type derivations. The unavailability of Weakening and Contraction as type derivation rules is manifest in a very simple way in meaning representations: every meaning placeholder or variable that appears N times[8] in a typing context Γ also appears free exactly N times on the right-hand side of the turnstile and every λ-bound variable occurs free exactly once in its binder's scope.

$$\begin{aligned} tom : e,\ like : e \multimap e \multimap t &\quad \nvdash like\ tom\ jerry : t \\ tom : e,\ like : e \multimap e \multimap t &\quad \nvdash like\ tom\ tom : t \\ tom : e,\ like : e \multimap e \multimap t,\ jerry : e,\ cry : e \multimap t &\quad \nvdash like\ tom\ jerry : t \\ tom : e,\ like : e \multimap e \multimap t,\ jerry : e &\quad \nvdash like\ tom\ tom : t \end{aligned} \tag{14}$$

The system based on $(\otimes, \multimap)$ propositional logic system rejects all four typing judgements of (14) whereas its intuitionistic counterpart would only prohibit the intuitionistic version of the first. The semantic type system Montague used was very similar to the one we have already rejected for meaning assembly. However, he did not use that type system to drive the derivation of semantic forms but a methodology based on the constituent structure of a sentence.

TLG lies somewhere between Montague's approach and Glue. Both TLG and Glue offer an elegant alternative to rule-by-rule semantic composition: type-driven semantic-form derivation. But TLG is more faithful to Montague's programme as in it meaning composition maintains a close relation to constituent syntactic structure. For its proponents, close ties between syntax and semantics are an advantage. Glue was designed to be part of a modular system, its role being to combine meanings. It can work with many syntax formalisms and many different analyses within them.

[7] It may help to think of '$\forall$' as meaning 'for any', rather than 'for all'.

[8] Because of the way the typing context is formed, N always equals 1 in practice.

TLG is a syntax formalism itself, not only a meaning composition framework, so it rejects Exchange on the grounds that word order encodes important non-redundent information. TLG's rejection of Exchange leads to order-sensitivity and a split of implication into a forwards and backwards variant.

Because our current linear type system has Exchange as one of its rules, it allows the derivation of a semantic form for (16) from the semantic contributions of (15). After all, viewed as multisets (which is what Exchange allows) the typing contexts of the two sentences are exactly the same. A linear type system with e and t as its only base types does not distinguish between subject and object entities, between verb phrases and nouns, and so on. Constituent structure as well as functional structure can provide information that avoids such problems. Following the original development of Glue, we will see how a linear type-system can be used for a programme of compositionality based on LFG functional structure.

Jerry likes Tom. (15)

$$f: \left[\begin{array}{l} \textsc{pred}\ \text{`LIKE'} \\ \textsc{subj}\ \ s : \left[\ \textsc{pred}\ \text{`JERRY'}\ \right] \\ \textsc{obj}\ \ \ o : \left[\ \textsc{pred}\ \text{`TOM'}\ \right] \end{array}\right].$$

Reading 1
$jerry : e, like : e \otimes e \multimap t, \qquad tom : e \vdash like(jerry, tom) : t.$
$*$*Reading 2*
$jerry : e, like : e \otimes e \multimap t, \qquad tom : e \vdash like(tom, jerry) : t.$

Tom likes Jerry. (16)

Our simple linear type system captures the requirement that each semantic contribution is used exactly once. What we need to do now is incorporate functional information into the base types. For that purpose, we will start using f-structure labels such as o, s and f as seen in the f-structure for (15). Note that an f-structure label captures the function of the semantic contribution in the sentence in absolute terms. This is important because when dealing with complex sentences specifying that a semantic contribution has, for example, the SUBJect function is not enough. Different clauses may have different subjects, but the f-structure label uniquely identifies the subject in question. We will briefly consider two alternative ways of using f-structure labels in order to capture functional information in the base types:

1. We can make a move from propositional to predicate logic in our type system. The types e and t are no longer base types (propositions), but base-type constructors of arity one (single-argument predicates) taking a label in the f-structure as their argument. A base-type constructor with all its arguments is a base type. Thus $jerry : e(s)$ is the subject entity, $tom : e(o)$ is the object entity of sentence f and $like : e(s) \otimes e(o) \multimap t(f)$ is a function from sentence f's subject and an object entities pair to its truth value.

2. We can keep the propositional type system and use f-structure labels directly as base types, thus encoding functional roles but at the cost of losing the distinction between *e* and *t* values. The second proposal is simple but unfortunately inadequate.

Propositional logic does not allow quantification but predicate logic does. As we shall soon see, quantification over labels is required in the types of certain kinds of semantic contribution. In the first proposal, support for quantification comes with the system, whereas in the second it is not supported due to its propositional nature. That fact alone is a perfectly valid reason to reject the second proposal in favour of first-order Glue. G3 with labels as parameters of base-type constructors supports quantification over labels, exactly as required. System F, to which the designers of G2 resorted in order to allow quantification over types (since labels are types in G2), supports higher-order quantification. However, higher-order quantification was found to be unsuitable. It was restricted. But even that was not enough. Inability to distinguish between entity and truth values also caused problems. These were solved by means of an extension of the system that made it support two sorts, *e* and *t*. G2, the end result of all these ad- hoc extensions and restrictions to the second proposal, is unnecessarily complicated. All the problems with G2 started with the decision to treat labels as base types. These matters are discussed in more detail in Sect. 4. We will continue following the first proposal (G3).

Fig. 2 First-Order Glue inference rules

$$\frac{N : T,\ \Gamma,\ N' : T',\ \Gamma' \vdash E : T''}{N' : T',\ \Gamma,\ N : T,\ \Gamma' \vdash E : T''}\ (Exchange)$$

$$N : T \vdash N : T\ (Axiom)$$

$$\frac{\Gamma, X : T \vdash E : T'}{\Gamma \vdash \lambda X.E : T \multimap T'}\ (\multimap Intro.)$$

$$\frac{\Gamma \vdash E : T' \multimap T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash E\ E' : T}\ (\multimap Elim.)$$

$$\frac{\Gamma \vdash E : T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash (E, E') : T \otimes T'}\ (\otimes Intro.)$$

$$\frac{\Gamma, X : T, X' : T' \vdash E : T'' \quad \Gamma' \vdash E' : T \otimes T'}{\Gamma, \Gamma' \vdash \mathbf{let}\ (X, X') = E'\ \mathbf{in}\ E : T''}\ (\otimes Elim.)$$

$$\frac{\Gamma \vdash E : T}{\Gamma \vdash E : \forall V.T}\ [where\ V \notin FV(Types(\Gamma))]\ (\forall Intro.)$$

$$\frac{\Gamma \vdash E : \forall V.T}{\Gamma \vdash E : T[V := L]}\ (\forall Elim.)$$

Note *The inference rules of G3 appear to be identical to those of System F* (*Fig. 3*)*. The two differences are:* (1) *the syntax of types* (*only in G3 do base types take arguments, i.e. constants, variables, or complex expressions also involving functions*) *and* (2) *quantification* (*it involves types in System F, but only base-type constructor arguments in G3*).

Notational conventions *The standard syntactic conventions of linear logic formulas apply: the longest subformula parse is preferred, '⊗' binds tighter than '⊸', '⊗' is left-associative and '⊸' is right-associative. Also:*

1. Lowercase Greek letters are used for type-side variables and lowercase Roman letters for type-side constants.
2. The list of arguments of a base-type constructor are written as a subscript to it. This convention helps reduce the number of parentheses and makes for easier reading of complicated types. So, the admittedly rather lengthier $\forall\alpha.(e(s) \multimap t(s)) \multimap (e(s) \multimap t(\alpha)) \multimap t(\alpha)$, using this convention, becomes $\forall\alpha.(e_s \multimap t_s) \multimap (e_s \multimap t_\alpha) \multimap t_\alpha$, that is both shorter and more readable.
3. Types are written in *Prenex Normal Form* (i.e. 'quantifiers first' form). Since every first-order logic formula has a PNF equivalent, we can always do that. So, we prefer $\forall\alpha.(e_s \multimap t_s) \multimap (e_s \multimap t_\alpha) \multimap t_\alpha$ to the equivalent $(e_s \multimap t_s) \multimap \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha$.

With functional information incorporated into the base types of our first-order Glue system, only the correct reading for (15) is derivable. Below is the entire type derivation in which we can clearly see that the Exchange rule has a role to play in deriving the correct reading (2nd line from the bottom) and also that thanks to our label-sensitive types it is now impossible for Exchange to be used to derive the erroneous reading. The convention we will use from now on, that the antecedent in an linear typing sequent is a multiset, will replace the Exchange rule.

$$\dfrac{\dfrac{\dfrac{L \vdash l : e_s \otimes e_o \multimap t_f}{L \vdash l : e_s \otimes e_o \multimap t_f} \quad \dfrac{T \vdash t : e_s \quad J \vdash j : e_o}{T, J \vdash (t,j) : e_s \otimes e_o}}{L,\ T, J \vdash l(t,j) : t_f}}{T,\ L, J \vdash l(t,j) : t_f,} \tag{17}$$

$$\text{where} \quad L = l : e_s \otimes e_o \multimap t_f,\ T = t : e_s,\ J = j : e_o.$$

The proposed first-order $(e/1, t/1)$ G3 system solves the problems of an intuitionistic propositional (e,t) type system that allows the derivation of additional composite meaning representations corresponding to invalid readings. It also allows quantification over labels. We will now see why this is an essential feature of Glue.

Everyone smiles. (18)

$$f : \left[\begin{array}{l} \textsc{pred} \text{ 'SMILE'} \\ \\ \textsc{subj} \ s : \left[\ \textsc{pred} \text{ 'EVERYONE'} \ \right] \end{array}\right]$$

Typing context:

$$\Gamma = \begin{array}{l} \mathit{everyone} : \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha, \\ \mathit{smile} : e_s \multimap t_f \end{array}$$

Reading: $(\alpha = f)$

$$\Gamma \vdash \mathit{everyone\ smile} : t_f.$$

Readers not familiar with the Glue treatment of quantification in natural language may wonder why universal quantification over labels is used in this example. They

may expect that the type of *everyone* would be $(e_s \multimap t_f) \multimap t_f$. There is a reason why this approach is not taken: it works well for simple sentences like (18) and (19), but not for more complicated examples like (21). But this is not simply a matter of trial and error. The ingenious idea of Dalrymple et al. (1995a) of introducing universal quantification over labels into Glue is actually motivated by theoretical (type theoretic and linguistic) considerations just as much as it is by examples like (21).

The fundamental idea behind this analysis is that the potential for combination of quantifier noun phrases with simple predicates should not be any more or less restricted than that of simple entity noun phrases. Assuming we had a simple entity, say *fernando*, instead of *everyone* as the subject of (18), its type would be e_s. Therefore, it would be possible to combine such a value (as the predicate argument) with any predicate P of type $e_s \multimap t_\alpha$, irrespective of what this α label is, giving a t_α result. The following type-lifting judgement is a concise expression of the same fact:

$$\mathit{fernando} : e_s \vdash \lambda P.\, P\, \mathit{fernando} : \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha.$$

We would expect a quantifier noun phrase to behave like a type-lifted simple entity noun phrase. Consequently, the type of *everyone* in (18) should be $\forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha$, rather than $(e_s \multimap t_f) \multimap t_f$, as it should be able to combine with any predicate $e_s \multimap t_\alpha$ for any label α, the type of the combination being t_α. So, $\mathit{everyone} : \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha$ will exhibit analogous behaviour to that of $\mathit{fernando} : e_s$ with respect to combining with any predicate $P : e_s \multimap t_\alpha$ irrespective of what α is.

Note that the presented analysis of complex noun phrases avoids LFG-specific solutions that might also have been able to give the expected range of readings even in complicated examples. Dalrymple et al. (1995a) not only provided a very powerful and elegant analysis but also one that does not tie Glue to LFG, thus paving the way for Glue to be used with a variety of syntax formalisms.

The beauty of the treatment of scope ambiguity in Glue is that nothing special is needed to account for the phenomenon. The types that were needed to get the readings of simple unambiguous sentences work just as well for more complex and ambiguous ones.

Everyone likes someone (19)

$$f : \left[\begin{array}{l} \text{PRED 'LIKE'} \\ \text{SUBJ } s : \left[\text{ PRED 'EVERYONE' }\right] \\ \text{OBJ } \quad o : \left[\text{ PRED 'SOMEONE' }\right] \end{array}\right]$$

Typing context:

$$\Gamma = \begin{array}{l} \mathit{everyone} : \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha, \\ \mathit{like} : e_s \otimes e_o \multimap t_f, \\ \mathit{someone} : \forall\beta.(e_o \multimap t_\beta) \multimap t_\beta. \end{array}$$

First reading: $(\alpha = \beta = f)$

$$\Gamma \vdash \mathit{everyone}\ \lambda x.\, \mathit{someone}\ \lambda y.\, \mathit{like}\ (x, y) : t_f.$$

Second reading: $(\alpha = \beta = f)$

$$\Gamma \vdash \textit{someone}\ \lambda y.\textit{everyone}\ \lambda x.\textit{like}\ (x, y) : t_f.$$

The reader may have noticed that each type judgement giving a Glue-derived reading is accompanied by the instantiations of all type-side variables in the typing context. This is meant to help appreciate the role of quantification in obtaining each reading and its overall role in the type system. Note that since both α and β were instantiated to f anyway, scope alteration in (19) did not rely on Glue's support of universal quantification over labels. Indeed, in our simple examples, no variable was ever instantiated to anything other than the outermost sentence label. If it did turn out to be the case that first-order quantification was strictly-speaking unnecessary in Glue analyses, its central place within the Glue framework would be hard to justify. That would not make analyses, such as the one for complex noun phrases, using it wrong, but it could mean that the first-order Glue system is unnecessarily powerful. This is not the case though. For instance, first-order quantification over labels plays a role in obtaining the readings of (21).

For now, let us consider a simpler example demonstrating the G3 analysis of generalised quantifiers and common nouns we will be using. This analysis avoids semantic projections which were invented for the corresponding analysis in earlier versions of Glue. This will pave the way for both the relevant discussion in Sect. 4 and the analysis of (21).

Every child plays. (20)

$$f : \left[\begin{array}{l} \textsc{pred}\ \text{`PLAY'} \\ \textsc{subj}\ s : \left[\begin{array}{l} \textsc{spec}\ \text{`EVERY'} \\ \textsc{pred}\ \text{`CHILD'} \end{array}\right] \end{array}\right]$$

Typing context:

$$\Gamma = \begin{array}{l} \textit{every} : \forall\alpha.(e_s \multimap t_s) \multimap (e_s \multimap t_\alpha) \multimap t_\alpha, \\ \textit{child} : e_s \multimap t_s, \\ \textit{play} : e_s \multimap t_f. \end{array}$$

Reading: $(\alpha = f)$

$$\Gamma \vdash \textit{every child play} : t_f.$$

A generalised quantifier (*every*) and a restrictor (*child*) are expected to combine and behave exactly like a quantifier. The restrictor is the argument of the generalised quantifier and its type $(e_s \multimap t_s)$[9] is the later's argument type. The return type of the generalised quantifier then must be the type of a quantifier $(\forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha)$. The type assigned to *every*, namely $\forall\alpha.(e_s \multimap t_s) \multimap ((e_s \multimap t_\alpha) \multimap t_\alpha)$, is simply $(e_s \multimap t_s) \multimap (\forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha)$ converted to Prenex Normal Form.

[9] In Sect. 4, we will see a number of alternative analyses, but for now we will use one in which (with reference to (20)) the restrictor and the body predicates take the same type of argument (e_s) but differ in that they return a value of type t_s and t_f, respectively.

Every representative of a firm took a sample. (21)

$$f:\left[\begin{array}{ll}\text{PRED}\ \text{`TOOK'} & \\ \text{SUBJ}\ s:\left[\begin{array}{ll}\text{SPEC} & \text{`EVERY'}\\ \text{PRED} & \text{`REPRESENTATIVE'}\\ \text{OBL}_{\text{OF}}\ r:\left[\begin{array}{ll}\text{SPEC} & \text{`A'}\\ \text{PRED} & \text{`FIRM'}\end{array}\right]\end{array}\right] & \\ \text{OBJ}\quad o:\left[\begin{array}{ll}\text{SPEC} & \text{`A'}\\ \text{PRED} & \text{`SAMPLE'}\end{array}\right] & \end{array}\right]$$

Typing Context:

$$\Gamma = \begin{array}{l} \mathit{every} : \forall\alpha.(e_s \multimap t_s) \multimap (e_s \multimap t_\alpha) \multimap t_\alpha,\\ \mathit{rep} : e_r \multimap (e_s \multimap t_s),\\ a_1 : \forall\beta.(e_r \multimap t_r) \multimap (e_r \multimap t_\beta) \multimap t_\beta,\\ \mathit{firm} : e_r \multimap t_r,\\ \mathit{take} : e_s \otimes e_o \multimap t_f,\\ a_2 : \forall\gamma.(e_o \multimap t_o) \multimap (e_o \multimap t_\gamma) \multimap t_\gamma,\\ \mathit{sample} : e_o \multimap t_o. \end{array}$$

First reading: $(\alpha = f,\ \beta = s,\ \gamma = f)$,
$\Gamma \vdash \mathit{every}\ (\lambda x.a_1\ \mathit{firm}\ \lambda y.\mathit{rep}\ y\ x)\ \lambda x.a_2\ \mathit{sample}\ \lambda z.\mathit{take}(x, z) : t_f.$

Second reading: $(\alpha = f,\ \beta = s,\ \gamma = f)$,
$\Gamma \vdash a_2\ \mathit{sample}\ \lambda z.\mathit{every}\ (\lambda x.a_1\ \mathit{firm}\ \lambda y.\mathit{rep}\ y\ x)\ \lambda x.\mathit{take}(x, z) : t_f.$

Third reading: $(\alpha = f,\ \beta = f,\ \gamma = f)$,
$\Gamma \vdash a_2\ \mathit{sample}\ \lambda z.a_1\mathit{firm}\ \lambda y.\mathit{every}\ (\mathit{rep}\ y)\ \lambda x.\mathit{take}(x, z) : t_f.$

Fourth reading: $(\alpha = f,\ \beta = f,\ \gamma = f)$,
$\Gamma \vdash a_1\ \mathit{firm}\ \lambda y.\mathit{every}\ (\mathit{rep}\ y)\ \lambda x.a_2\ \mathit{sample}\ \lambda z.\mathit{take}(x, z) : t_f.$

Fifth reading: $(\alpha = f,\ \beta = f,\ \gamma = f)$,
$\Gamma \vdash a_1\ \mathit{firm}\ \lambda y.a_2\ \mathit{sample}\ \lambda z.\mathit{every}\ (\mathit{rep}\ y)\ \lambda x.\mathit{take}(x, z) : t_f.$

A relational noun such as 'representative' needs an entity in order to have an ordinary noun (i.e. predicate) meaning. To say of someone that they are a representative begs the question of whom or of what they are a representative. So, the semantic contribution of 'representative', *rep*, has the type $e_r \multimap (e_s \multimap t_s)$ in this example, *rep y* (where *y* is the entity being represented) is a predicate of type $e_s \multimap t_s$, and *rep y x* is the statement (of type t_s) '*x* is a representative of *y*'.

The Glue analysis of quantification correctly predicts the five readings of (21). This would not have been possible if β was not free to be instantiated to either *s* or *f*. The first instantiation gives the first two readings and the latter the remaining three.

This completes the discussion which aimed to motivate the design of G3, its resource sensitivity, its label sensitivity and the need for the later to come with support for first-order universal quantification over labels. All that is obtained by adopting a first-order linear type system. Next, we will see how first-order Glue relates to its predecessor.

3 Relating first-order Glue to its predecessor

G2 can be seen as a somewhat impoverished version of G3. This is rather puzzling considering that G2 is based on a higher-order system and that, on top of that, it has

been extended to a two-sorted system whereas G3 is a humble, plain and simple first-order system. We will see that it can be proved that for all its System F (Girard, 1989)[10] heritage, G2, the current Glue system is actually closer to the proposed first-order system than to F, despite notational appearances.

The key to the puzzle is the observation that whereas quantification over types in F is beyond the encoding capacity of a first-order system, quantification over base types only is not. This is first demonstrated for F1, a system simpler than G2, but inadequate as a potential Glue system. This is because when labels are used as base types in F1, there is no mechanism that can be used to distinguish between truth and entity types.

Fig. 3 (linear) System F inference rules

$$\frac{N : T,\ \Gamma,\ N' : T',\ \Gamma' \vdash E : T''}{N' : T',\ \Gamma,\ N : T,\ \Gamma' \vdash E : T''}\ (Exchange)$$

$$N : T \vdash N : T\ (Axiom)$$

$$\frac{\Gamma, X : T \vdash E : T'}{\Gamma \vdash \lambda X.E : T \multimap T'}\ (\multimap Intro.)$$

$$\frac{\Gamma \vdash E : T' \multimap T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash E\ E' : T}\ (\multimap Elim.)$$

$$\frac{\Gamma \vdash E : T \quad \Gamma' \vdash E' : T'}{\Gamma, \Gamma' \vdash (E, E') : T \otimes T'}\ (\otimes Intro.)$$

$$\frac{\Gamma, X : T, X' : T' \vdash E : T'' \quad \Gamma' \vdash E' : T \otimes T'}{\Gamma, \Gamma' \vdash \mathbf{let}\ (X, X') = E'\ \mathbf{in}\ E : T''}\ (\otimes Elim.)$$

$$\frac{\Gamma \vdash E : T}{\Gamma \vdash E : \forall V.T}\ [where\ V \notin FV(Types(\Gamma))]\ (\forall Intro.)$$

$$\frac{\Gamma \vdash E : \forall V.T}{\Gamma \vdash E : T[V := T']}\ (\forall Elim.)$$

Definition 1 *F1 is F with the restriction on type-inference rules that a type variable can only be instantiated to a constant base type or another variable.*

Definition 2 *G3/t is G3 restricted so that it only includes typed terms in which no base-type constructor other than $t/1$ is used and an argument of a base-type constructor can only be atomic* (*a variable or a constant, but not the result of some function*).

Definition 3 *The function ω_{F1} that maps F1 typed terms to typed terms in G3/t is defined as follows:*

$$\omega_{F1}(\Lambda : T) = \Lambda : \tau_{F1}(T),$$

[10] Following Dalrymple et al. (1997a), we are assuming a linear version of System F that supports $\otimes$ and has a simplified syntax for the $\forall$ rules. This is not exactly the System F as presented by Girard (1989).

where

$$\begin{aligned}\tau_{F1}(A) &= t(A),\\ \tau_{F1}(T \multimap T') &= \tau_{F1}(T) \multimap \tau_{F1}(T'),\\ \tau_{F1}(T \otimes T') &= \tau_{F1}(T) \otimes \tau_{F1}(T'),\\ \tau_{F1}(\forall X.T) &= \forall X.\tau_{F1}(T).\end{aligned}$$

Theorem 1 ω_{F1} is a bijection.

Proof Given any two different F1 typed terms their ω_{F1} images will be different and every G3/*t* typed term is an ω_{F1} image of an F1 typed term. This is obvious from the definition of ω_{F1}. □

Definition 4 *Two type systems, T1 and T2, are directly interchangeable, if and only if there is a bijective function $\mathcal{G}$ from T1 to T2 type judgements such that for any T1 type judgement J and (possibly empty) list of type judgements $J_1, \ldots, J_n$ there is an inference rule in T1 by which J can be derived in T1 given $J_1, \ldots, J_n$ if and only if there is an inference rule in T2 by which $\mathcal{G}(J)$ can be derived in T2 given $\mathcal{G}(J_1), \ldots, \mathcal{G}(J_n)$, with the same, if any, side-conditions.*

Theorem 2 F1 and G3/*t* are directly interchangeable.

Proof Trivial. The bijection used is:

$$\mathcal{G}_1(A_1 : T_1, \ldots, A_n : T_n \vdash \Lambda : T) = \omega_{F1}(A_1 : T_1), \ldots, \omega_{F1}(A_n : T_n) \vdash \omega_{F1}(\Lambda : T).$$ □

That F1 and G3/*t* are directly interchangeable is a very strong result. It basically means that they are indeed different only in notation. The following derivation given in both systems is a simple demonstration of that:

$$\dfrac{\dfrac{q : \forall\alpha.(s \multimap \alpha) \multimap \alpha \vdash q : \forall\alpha.(s \multimap \alpha) \multimap \alpha}{q : \forall\alpha.(s \multimap \alpha) \multimap \alpha \vdash q : (s \multimap f) \multimap f} \; p : s \multimap f \vdash p : s \multimap f}{q : \forall\alpha.(s \multimap \alpha) \multimap \alpha,\ p : s \multimap f \vdash q\ p : f} \tag{22}$$

$$\dfrac{\dfrac{q : \forall\alpha.(t(s) \multimap t(\alpha)) \multimap t(\alpha) \vdash q : \forall\alpha.(t(s) \multimap t(\alpha)) \multimap t(\alpha)}{q : \forall\alpha.(t(s) \multimap t(\alpha)) \multimap t(\alpha) \vdash q : (t(s) \multimap t(f)) \multimap t(f)} \; p : t(s) \multimap t(f) \vdash p : t(s) \multimap t(f).}{q : \forall\alpha.(t(s) \multimap t(\alpha)) \multimap t(\alpha),\ p : t(s) \multimap t(f) \vdash q\ p : t(f)} \tag{23}$$

We can now use our experience with F1 to relate G2 to G3. G2 is based on System F extended to two sorts, but the restriction placed upon quantification, which only allows it to work with base types of the *t* sort, means that, like F1, it is just a first-order system in disguise.

Definition 5 *G2 is F extended to two sorts, t and e with the restriction that a type variable can only be of the t-sort and can only be instantiated to a t-sort constant base type or another variable.*

Definition 6 *G3/t[e] is G3 restricted so that it only includes typed terms in which no base type constructors other than $t/1$ and $e/1$ are used, a base-type constructor argument can only be atomic (a variable or a constant, but not the result of some function) and cannot be used with both $t/1$ and $e/1$ type constructors, and, finally, variables cannot be arguments to $e/1$ type constructors.*

Definition 7 *The function ω that maps G2 typed terms to typed terms in G3/t[e] is defined as follows:*

$$\omega(\Lambda : T) = \Lambda : \tau(T),$$

where

$$\begin{aligned}
\tau(A_e) &= e(A),\\
\tau(A_t) &= t(A),\\
\tau(T \multimap T') &= \tau(T) \multimap \tau(T'),\\
\tau(T \otimes T') &= \tau(T) \otimes \tau(T'),\\
\tau(\forall X.T) &= \forall X.\tau(T).
\end{aligned}$$

Theorem 3 ω is a bijection.

Proof Given any two different G2 typed terms their ω images will be different and every G3/t[e] typed term is an ω image of a G2 typed term. □

Theorem 4 G2 and G3/t[e] are directly interchangeable.

Proof Trivial. The bijection used is:

$$\mathcal{G}(A_1 : T_1, \ldots, A_n : T_n \vdash \Lambda : T) = \omega(A_1 : T_1), \ldots, \omega(A_n : T_n) \vdash \omega(\Lambda : T).$$

□

So, G2 and G3/t[e] are indeed different only in notation. To illustrate, we turn to what could have been a derivation of the composite meaning of a sentence with a quantifier q and an intransitive verb p:

$$\dfrac{\dfrac{q : \forall\alpha_t.(s_e \multimap \alpha_t) \multimap \alpha_t \vdash q : \forall\alpha_t.(s_e \multimap \alpha_t) \multimap \alpha_t}{q : \forall\alpha_t.(s_e \multimap \alpha_t) \multimap \alpha_t \vdash q : (s_e \multimap f_t) \multimap f_t} \quad p : s_e \multimap f_t \vdash p : s_e \multimap f_t}{q : \forall\alpha_t.(s_e \multimap \alpha_t) \multimap \alpha_t,\ p : s_e \multimap f_t \vdash q\ p : f_t} \tag{24}$$

$$\dfrac{\dfrac{q : \forall\alpha.(e(s) \multimap t(\alpha)) \multimap t(\alpha) \vdash q : \forall\alpha.(e(s) \multimap t(\alpha)) \multimap t(\alpha)}{q : \forall\alpha.(e(s) \multimap t(\alpha)) \multimap t(\alpha) \vdash q : (e(s) \multimap t(f)) \multimap t(f)} \quad p : e(s) \multimap t(f) \vdash p : e(s) \multimap t(f)}{q : \forall\alpha.(e(s) \multimap t(\alpha)) \multimap t(\alpha),\ p : e(s) \multimap t(f) \vdash q\ p : t(f)} \tag{25}$$

Given the strength of our result regarding G2 and G3/t[e], it is fairly obvious that to convert a G2 implementation to a G3/t[e] implementation, all one needs to do is change the syntax of types. It is clear that G3/t[e], therefore also G2, is a restricted fragment of G3. The definition of G3/t[e] was carefully chosen to reflect the properties of G2. It tells us exactly what G2 amounts to in the realm of First-Order Glue.

Seeing G2 as a restricted version of G3 we might wonder if any of the restrictions are necessary. The easy answer is 'none'. G3 in its entirety does not hide any traps for our programme of computational linguistics. It is up to its users to come up with correct analyses and it gives them more tools to do this easily and elegantly (user-chosen base-type constructors, not necessarily of arity 1, functions on labels and a better use of labels in forming base types). Section 4 discusses the design of G2 from a different perspective and answers many similar questions.

4 On the design of G2 and G3

Probably, the most 'advertised' feature of Glue is its resource sensitivity. This discussion will start boldly, that is by questioning the necessity of one of the fundamental features of Glue. Some motivation for resource sensitivity was given in Sect. 2. There, two pairs of sentences, (11)&(10) and (12)&(13), were used to show the kind of problems Weakening and Contraction may, respectively, cause. There are two things to note. One is that both of these examples involved modification. The other is that at that point a simple (e,t) propositional type system inadequate for our purposes was still being used. It is reasonable to ask if a better type system with an appropriate analysis for modification can solve the problems Weakening and Contraction can cause, without being resource sensitive.

Indeed, label sensitivity coupled with a different analysis of modification can render the Weakening and Contraction rules inapplicable even when they are part of the logic. Here is an example of a typing context for (10) using a non-standard analysis of modification[11] that disallows throwing away or duplicating modifiers:[12]

$$\mathit{mary} : e_s, \quad \mathit{sing} : e_s \to t_{f'}, \quad \mathit{happily} : (e_s \to t_{f'}) \to (e_s \to t_f).$$

In this analysis, the use of the intermediate label f' guarantees that one has to apply the modifier to the verb in order to then combine the result with the subject entity. Multiple modifiers will only combine in the prescribed order. The order-sensitive analysis avoids the problems its order-insensitive standard counterpart has in the treatment of multiple modification ('*ancient fake golden crown*') and its interaction with modifier modification ('*obviously wrong third answer*').

Even though, this analysis of modification shows that label sensitivity may be used to provide the effects of resource sensitivity, there are reasons why the latter is explicitly featured in Glue. First, Glue, unlike the intuitionistic type system used above, allows both analyses of modification exactly because it is resource sensitive. Furthermore, eliminating two structural inference rules, as Glue (linear logic) does, reduces the complexity of the system, whereas having them and trying hard to disallow their application has no true advantage. A third argument in favour of resource sensitivity in Glue is that it is an essential principle of compositional semantics; that Glue formally enforces it is one of its most attractive features. Finally, resource sensitivity *is* necessary. Weakening and Contraction combined would be able to make the following invalid judgement possible in Glue:

$$\mathit{nobody} : \forall\alpha.(e_s \multimap t_\alpha) \multimap t_\alpha,\ \mathit{cry} : e_s \multimap t_f \nvdash \mathit{nobody}\ \lambda x.\ \mathit{nobody}\ \mathit{cry} : t_f.$$

Label sensitivity is not a feature of a type system as such, but of a formal system for meaning assembly of semantic contributions associated with syntactic (or related) structures with identifying labels. Those labels are used to form the base types in G2 and G3 (the two Glue systems that are type systems), albeit in somewhat different ways.

The decision to use labels as base types in G2 is the source of most of its unnecessary complexity, simple as the idea may seem at first. In G3, labels are parameters of

[11] A. Andrews, Glue logic, projections and modifiers (2003, Unpublished paper) provides a fully worked-out analysis very similar to this one.

[12] A first-order intuitionistic type system (allowing Weakening and Contraction) with our parameters-as-subscripts convention is assumed.

base-type constructors (predicates) and first-order quantification is readily available, but if labels are themselves types, quantification over types is needed instead. This leads to a leap from a simple system based on propositional logic to a second-order system such as System F.

G2 is based on System F but it is not System F. The most important difference is that higher-order quantification, the hallmark of F (second-order λ-calculus), is prohibited in G2. To see why this is so, we may consider the following linear System F typing context:

$$\begin{aligned} \mathit{everyone} &: \forall A.(s \multimap A) \multimap A, \\ \mathit{love} &: s \otimes o \multimap f, \\ \mathit{someone} &: \forall B.(o \multimap B) \multimap B. \end{aligned}$$

Since

$$\mathit{love} : s \otimes o \multimap f \vdash \lambda x.\lambda y.\mathit{love}\,(x, y) : s \multimap o \multimap f,$$

higher-order universal quantification allows the following:

$$\begin{aligned} \mathit{everyone} &: \forall A.(s \multimap A) \multimap A, \\ \mathit{love} &: s \otimes o \multimap f \end{aligned} \vdash \mathit{everyone}\ \lambda x.\lambda y.\mathit{love}\,(x, y) : o \multimap f.$$

What we wanted was to allow A and B to range over truth base types. What we got instead was A instantiated to a type of a function from entities to truth values ($o \multimap f$). Instantiating B to f we get:

$$\begin{aligned} \mathit{everyone} &: \forall A.(s \multimap A) \multimap A, \\ \mathit{love} &: s \otimes o \multimap f, \\ \mathit{someone} &: \forall B.(o \multimap B) \multimap B \end{aligned} \qquad \vdash \mathit{someone}\ (\mathit{everyone}\ \lambda x.\lambda y.\mathit{love}\,(x, y)) : f.$$

So the problem is not that F can not derive the two readings of sentence (19). Of course it can. A universally quantified variable can be instantiated to any type and labels are types. However, when it is meant to be instantiated to a label corresponding to a truth type but it is instantiated to a type corresponding to a predicate instead, the corresponding semantic expression will be ill-formed. For illustration purposes [assuming a combination of Glue with a compositional variant of DRT as the semantic representation language (van Genabith & Crouch, 1997; Kokkonidis, 2005)], the above 'reading' is contrasted with the reading of (19) where *someone* takes wide scope:

SYSTEM F $\quad \mathit{someone}\ (\mathit{everyone}\ \lambda x.\lambda y.\mathit{love}(x, y))$

DRT

$$* \left(\begin{array}{|c|} \hline a \\ \hline \mathit{person}(a) \\ \hline \end{array} \Rightarrow \begin{array}{|c|} \hline \\ \hline \lambda y.\mathit{love}(a, y) \\ \hline \end{array} \right)(b) \quad \text{within} \quad \begin{array}{|c|} \hline b \\ \hline \mathit{person}(b) \\ \hline \end{array}$$

SYSTEM F & GLUE $\quad \mathit{someone}\ \lambda y.\mathit{everyone}\ \lambda x.\mathit{love}(x, y)$

DRT

$$\begin{array}{|c|} \hline b \\ \hline \mathit{person}(b) \\ \hline \begin{array}{|c|} \hline a \\ \hline \mathit{person}(a) \\ \hline \end{array} \Rightarrow \begin{array}{|c|} \hline \\ \hline \mathit{love}(a, b) \\ \hline \end{array} \\ \hline \end{array}$$

A formal system for compositionality that gives us as sentence readings λ-calculus terms that correspond to ill-formed expressions in the meaning language is not what we want. The main problem with F is that its notion of quantification over types does not distinguish between a simple base type and a complex type. We have shown that F and G2 may look similar but in fact behave rather differently. Since what we wanted was to quantify over (truth) labels which G2 has taken to be base types, we may wonder if F1, a linear F with quantification restricted to base types (see Sect. 3), is the solution. As it turns out, the designers of G2 (and G1) have also considered the possibility of not having two sorts (M. Dalrymple, 2004, personal communication). However, they discovered that restricting quantification to base types was not enough. For example, given the typing context

$$\mathit{everyone} : \forall A.(s \multimap A) \multimap A,\ \mathit{sleep} : s \multimap f,$$

nothing prohibits an instantiation of A to s (a base type) in F1. Without using up any resources (semantic contributions in the typing context) we may obtain the identity function for entities of type s:

$$\vdash \lambda x.\, x : s \multimap s.$$

Combining the quantifier and the identity function we get a subject entity:

$$\mathit{everyone} : \forall A.(s \multimap A) \multimap A \vdash \mathit{everyone}\ \lambda x.\, x : s.$$

And to complete the problematic derivation, we use that as the argument of *sleep* to get the following typing judgement:

$$\mathit{everyone} : \forall A.(s \multimap A) \multimap A,\ \mathit{cry} : s \multimap f \vdash \mathit{cry}\ (\mathit{everyone}\ \lambda x.\, x) : f.$$

Because it can not distinguish between entity and statement types, F1 is not suitable to be used as a type system driving meaning assembly.

Restricting the complexity of the types to which a variable can be instantiated is not enough. The designers of G2, aware of the problems of F1, extended a linear F to two sorts (t and e), thus obtaining a distinction between truth and entity types, and only allowed quantification over base types, actually only of the t sort as they saw no practical motivation for expressing quantification over e base types.

This discussion hopefully sheds some light into why G2 is as complicated as it is. The problems started because labels as base types (or propositions in G2's predecessors) seemed like a good idea, but as it turns out it is not. G3 has a first-order type system where labels are mere arguments to base-type constructors. The claim made here is that this is exactly how Glue should have been designed in the first place. The advantages become clear when we consider how G3 avoids the problems G2 was designed to solve, not by a series of patches to an initial inadequate design, but by being based on the right formal system for the given task right from the start.

- Whereas the *designers* of G2 had to extend a version of System F to two sorts to avoid 'readings' corresponding to ill-formed meaning expressions, *users* of G3 are free to choose any number of base-type constructors; it just happens that using $t/1$ and $e/1$ is a fairly natural choice that also avoids that and other such problems.
- Glue analyses require support for label sensitivity and quantification over labels from the formal system. Propositional logic does not support quantification. Quantification over types as found in System F is too permissive; it had to be restricted to base types in G2. G3 supports label sensitivity by allowing labels as arguments

of the base type constructors. It also supports first-order quantification over them, exactly the kind of quantification required.

G3 and G2 represent different approaches to the problem of finding the ideal type-theoretic Glue framework. G3 is a general-purpose type system which we have discovered to be suitable for meaning composition when used with base-type constructors $e/1$ and $t/1$. G2 is a special-purpose system specifically designed for meaning composition: the e and t sorts are part of the system and the restricted version of second-order quantification has a very specific purpose to serve which also explains why it is only supported for the sort for which it is needed (t). A formal system designed for a specific purpose may have certain advantages over a general-purpose one. But G2 is neither simpler than G3, nor does its design reveal the nature of the problem better than $(e/1, t/1)$ G3; arguably, the opposite is true. That one can see in the design of G2 that quantification over e types is considered unnecessary, is perhaps its only advantage in that respect.

Next we turn our attention to the syntax-Glue interface. That interface has traditionally been through 'semantic projections' in LFG. However, what 'semantic projections' means has changed over the years. In G0, a semantic projection of an f-structure was what its name implies: 'the semantic projection of an f-structure, written with a subscript σ, is a representation of the meaning of that f-structure' (Dalrymple, Lamping, and Saraswat, 1993). But in G1, 'semantic projections' became attribute-value structures merely associated with meaning via the $\rightsquigarrow$ relation: 'semantic projections can carry more information than just the association to the meaning for the corresponding f-structure' (Dalrymple et al., 1995a). One use of the internal structure of G1 semantic projections was in the treatment of noun phrases having a generalised quantifier and a common noun as the prominent constituents. The semantic projection of such a noun phrase would have a VARiable and a RESTRictor attribute. One may reasonably ask what the information these attributes held was. Well, they really are dummy attributes that hold no information whatsoever. At least in G1 they were associated with a meaning expression through the $\rightsquigarrow$ relation: VAR would be, say, associated with a universally quantified X and RESTR with, say, $R(X)$, where R is the predicate that corresponds to the meaning of the common noun.

In G2, the role of 'semantic projections' became even more insignificant: VAR and RESTR attributes are not there to be directly or indirectly associated with a meaning, nor to hold any kind of information, but only to form the Glue type of a semantic contribution. It is hard to justify the adjective 'semantic' for a structure that only has dummy attributes or no attributes at all and no direct relation to the meaning representation of an f-structure. It is hard to justify the existence of dummy attributes in the first place. Andrews (2004) directly attacks semantic projections, which he disposes of, but not having solved the problem of dummy attributes, he transfers them into f-structures, something the Glue designers wanted to avoid. Where to put the dummy attributes is a concern also when Glue is used with a syntactic framework other than LFG.

G3 being a richer system helps avoid dummy attributes, that arguably, having no informational content, have no role in *any* structure. The true problem they have been used to solve in G1 and G2 is that new base types are needed for some analyses.[13]

[13] Actually, it looks like there has been a formal oversight as equal (empty) structures in LFG would normally not be considered to be distinct (M. Dalrymple, 2005, personal communication). Details aside, the intention was always very clear, which may be why the formal problem has gone unnoticed.

G3 offers three different ways of dealing with this need. Providing a type-based alternative to dummy attributes opens the possibility for Glue without them, and also for LFG Glue without G2-style 'semantic' projections.

Every child (26)

f-structure *G2-style semantic projection*

$$s:\begin{bmatrix}\textsc{spec}\ q:\text{`EVERY'}\\ \textsc{pred}\ n:\text{`CHILD'}\end{bmatrix} \qquad s_\sigma:\begin{bmatrix}\textsc{var}\ v:[\,]\\ \textsc{restr}\ r:[\,]\end{bmatrix}$$

$$\Gamma = \mathit{every}:? \multimap (e_s \multimap t_\alpha),\ \mathit{child}:?$$

1. G3 is not restricted to two base-type constructors, $e/1$ and $t/1$. In a situation where we need new base types, we can introduce new base-type constructors. For example, some possibilities for the type of *child* include $var_s \multimap restr_s$, cn_s and $e_{s,c} \multimap t_s$.[14]
2. An individual in first-order logic may be a constant or a variable, or the result of a function on individuals. If label s is an individual, so are $var(s)$ and $restr(s)$. Without introducing new base-type constructors, we could give *child* the type $e_{\mathrm{var}(s)} \multimap t_{\mathrm{restr}(s)}$.
3. In G2, a label is either of the e or the t sort.[15] So L labels give L base types in G2. In G3, labels are potential arguments to base-type constructors, so C base-type constructors of arity 1 combined with L labels result in $L \times C$ distinct base types. For example, the analysis for common nouns used in this paper refers only to the NP f-structure label which is used to form the two base types in $e_s \multimap t_s$ (NP e - NP t). If we allow ourselves to also refer to the f-structure label of the noun phrase predicate (the common noun) we have three more possibilities as one can see below:

Analysis	G3	G2
NP e - NP t	$e_s \multimap t_s$	N/A
CN e - NP t	$e_n \multimap t_s$	N/A
NP e - CN t	$e_s \multimap t_n$	$s_e \multimap n_t$
CN e - CN t	$e_n \multimap t_n$	N/A

An unexpected discovery was that one of these four analyses can also be used in G2. There had always been a way of avoiding the dummy VAR and RESTR attributes in Glue, but it was never used. This discovery does not change the fact that in G3 there will always be more choices. This discussion was meant to show a path for moving beyond two sins of Glue's past: G2-style 'semantic' projections and dummy attributes. However, at the same time it was meant to show that G3 offers significantly more freedom to its users in creating base types. This freedom can and should be used for proposing elegant new analyses.

Let us now turn our attention to ways in which Glue has or could be simplified. Linear logic is very rich, but the tendency has been towards using the minimum fragment possible.

[14] Note that $e/1$ and $e/2$ are distinct base-type constructors.

[15] G2 inherited this restriction from G1. There, each semantic projections label was associated with a semantic expression, which meant it could either be an entity or a truth value, but not both.

To a large extend, whether some part of linear logic is to be considered part of Glue or not depends on whether researchers need it. An analysis of anaphora using '!' was considered by Dalrymple, Lamping, Pereira, and Saraswat (1999) but this never became popular so '!' is not usually considered to be a part of Glue. The status of '$\otimes$' is less clear. There seems to be no consensus among Glue researchers: some would prefer to see the Glue type-system restricted to the $(\multimap, \forall)$ fragment of first-order linear logic, while others use the tensor and argue in favour of retaining it.

First, let us consider analyses that use '$\otimes$'. In early Glue literature the tensor was used in the meaning constructors of verbs with multiple arguments. The well-known currying technique can be used to avoid this usage. For example, instead of $like : e_s \otimes e_o \multimap t_f$ one can use its curried alternative $like : e_s \multimap e_o \multimap t_f$. Indeed, it is the later form that is more common now. But currying can not always be used as a substitute for forming pairs of values. The formation of pairs features prominently in a number of analyses. The phenomenon which, thanks to a number of such analyses, has been associated with pair formation the most is anaphora. Dalrymple et al. (1999) proposed an analysis of anaphora according to which a pronoun produces an additional copy of its antecedent; the semantics of the pronoun is $\lambda x.(x, x)$ and its glue type is $e_a \multimap e_a \otimes e_o$, assuming it appears as noun phrase o and its antecedent is a. This is the resource duplication treatment of anaphora. There are two more analyses of anaphora where '$\otimes$' makes an appearance (Crouch & van Genabith, 1999; Dalrymple, 2001); both of them also involve additional non-standard extensions to Glue. Kokkonidis (2005) argues against all of the aforementioned analyses and their underlying idea of having Glue manage anaphoric contexts and proposes an approach (similar to that of van Genabith & Crouch, 1997) which assigns semantic composition to Glue (without the tensor), and anaphoric context management and resolution to DRT. But there are also analyses not involving anaphora where '$\otimes$' appears such as the theory of negative polarity licensing of Fry (1999) and the proposal Potts (2003) put forwards for treating conventional implicatures in Glue.

It is not always the case that there are alternative analyses in the literature to those using '$\otimes$'. It is interesting to see if there are systematic ways of eliminating '$\otimes$' from analyses that use it, while maintaining their properties. Currying can help in cases where a pair is used as an argument to a function; it is not applicable when the goal is to encode the construction of a pair of values. One way of trying to encode a product (x, y) of type $X \otimes Y$ in a tensor-less System F is by $\lambda f.f\ x\ y$ of type $\forall Z.(X \multimap Y \multimap Z) \multimap Z$. A tensor-less Glue approximation of that would have type $\forall \zeta.(X \multimap Y \multimap t_\zeta) \multimap t_\zeta$. For example, a tensor-free alternative to the resource duplication analysis of anaphora (Dalrymple et al., 1999) has $\lambda x.\lambda P.P\ x\ x$ as the semantics of a pronoun (instead of $\lambda x.(x, x)$) and $\forall \zeta.e_a \multimap (e_a \multimap e_o \multimap t_\zeta) \multimap t_\zeta$ as its Glue type (instead of $e_a \multimap e_a \otimes e_o$). This encoding of pairs in Glue may not be perfect[16] but because of the truth-centric nature of truth-valued semantics it is sufficient for most intents and purposes. Still it can be cumbersome to work with; one may well prefer to be able to represent pairs directly, rather than through higher-order functions. What simplifies the formal framework, does not necessarily simplify analyses using the framework. There are some analogies with trying to simplify a language by eliminating 'difficult' words that can be roughly expressed in simpler terms; the

[16] For example, while $p : e_x \otimes e_y$, $f : e_x \multimap e_y \multimap e_r \vdash \text{let } (x, y) = p \text{ in } f\ x\ y : e_r$, $p : e_x \otimes e_y$ thus encoded in $(\multimap, \forall)$ G3 has type $\forall \zeta.(e_x \multimap e_y \multimap t_\zeta) \multimap t_\zeta$, and so it cannot take $f : e_x \multimap e_y \multimap e_r$ as its argument nor can the two combine otherwise.

$$\frac{\Gamma \vdash E : T}{\Gamma \vdash \Lambda V.\, E : \forall V.\, T}\ [where\ V \notin FV(Types(\Gamma))] \quad (\forall Intro.)$$

$$\frac{\Gamma \vdash E : \forall V.T}{\Gamma \vdash E\ L : T[V := L]}\ (\forall Elim.)$$

Fig. 4 Alternative first-order Glue ∀-rules

lexicon is simplified but sentences involving the concepts the eliminated words were there to express may become long and clumsy.

There can be arguments in favour or against retaining '⊗' in G3. But as long as some researchers choose to use the tensor and others do not find a place for it in their work, it seems that there will be a $(\multimap, \otimes, \forall)$ and a $(\multimap, \forall)$ version of Glue.

We will now turn our attention to quantification in Glue. As first-order logic comes with two kinds, universal and existential, the omission of the latter kind of quantification, justified by the fact that there have been no analyses that make use of it, is the first noteworthy quantification-related simplification of the system.

While there is a Curry–Howard isomorphism between the version of first-order Glue with the alternative ∀-rules of Fig. 4, $G3_\Lambda$, and the $(\multimap, \otimes, \forall)$ fragment of first-order linear logic, the same is not true for G3 (Fig. 2). However, such an isomorphism can be established between G3 and derivations in a normal form disallowing unnecessary uses of the ∀-rules. The link between first-order Glue and irreducible derivations in its corresponding fragment of linear logic is interesting; G3 and $G3_\Lambda$ offer slightly different perspectives. However, for the intended uses and audience of Glue, the simplicity of G3 syntax that avoids cluttering terms with notation related more to types was preferred.

5 Conclusions

The Glue approach has been a breakthrough in syntax-semantics interface research in general and a particularly important development for LFG whence it originated. G2 is well established and a move to G3 should have particularly good supporting reasons. G3 does not rely on strange, however well motivated, restrictions and it is exactly what it appears to be. The standard Glue typing system has the look of a second-order system, borrowing the notation of System F, but due to the restrictions placed on quantification, it and a fragment of the proposed system are directly interchangeable. All existing G2-based work in the Glue literature can be transferred effortlessly to the proposed system. Moreover, First-Order Glue makes possible more elegant alternatives, and may help researchers develop analyses that would not have been accommodated very well by the old system. I have argued that a fragment of first-order linear logic is the ideal basis for the Glue approach and that the proposed system offers numerous advantages that will facilitate further research in semantic composition for natural languages. I believe the elegance and added power of First-Order Glue to be compelling reasons for acceptance of the proposed system as the next step in the evolution of Glue.

Acknowledgements I am grateful to Mary Dalrymple for a number of discussions during the last few years, that provided not only an invaluable insight into the development of the Glue approach, but also endless enthusiasm and encouragement. Thanks also to Avery Andrews, Ash Asudeh, Dick

Crouch, Nissim Francez, John Lamping, Ioana Luca and the two anonymous referees for commenting on earlier versions of this work. This work has been supported by AHRC grant 2005/118667.

References

Andrews, A. D. (2004). Glue logic versus spreading architecture in LFG. In C. Mostovsky (Ed.), *Proceedings of the 2003 Conference of the Australian Linguistics Society*. Newcastle, Australia.

Asudeh, A., & Crouch, R. (2001). Glue semantics: A general theory of meaning composition. Talk given at Stanford Semantics Fest 2, March 16, 2001. http://csli.publications.stanford.edu/LFG/7/lfg02asudehcrouch.pdf

Asudeh, A., & Crouch, R. (2002). Glue semantics for HPSG. In F. van Eynde, L. Hellan, & D. Beermann (Eds.), *Proceedings of the 8th International HPSG Conference*. Stanford: CSLI Publications.

Crouch, R., & van Genabith, J. (1999). Context change, underspecification, and the structure of Glue language derivations. In M. Dalrymple (Ed.), *Semantics and syntax in Lexical Functional Grammar: The resource logic approach*, Cambridge, MA: MIT Press.

Dalrymple, M. (Ed.). (1999). *Semantics and syntax in Lexical Functional Grammar: The resource logic approach*. Cambridge, MA: MIT Press.

Dalrymple, M. (2001). *Lexical Functional Grammar*, No. 42 in Syntax and Semantics Series. NewYork: Academic Press.

Dalrymple, M., Gupta, V., Pereira, F. C., & Saraswat, V. (1997a), Relating resource-based semantics to categorial semantics. In *Proceedings of the Fifth Meeting on Mathematics of Language* (*MOL5*). Saarbrücken, Germany: Schloss Dagstuhl. An updated version was printed in M. Dalrymple (Ed.), *Semantics and syntax in Lexical Functional Grammar: The resource logic approach*. Cambridge, MA: MIT Press.

Dalrymple, M., Lamping, J., Pereira F. C., & Saraswat, V. (1995a). A deductive account of quantification in LFG. In K. Makoto, C. J. Pinón, & H. de Swart (Eds.), *Quantifiers, deduction and context*. Stanford, CA: Center for the Study of Language and Information.

Dalrymple, M., Lamping, J., Pereira, F. C., & Saraswat, V. (1995b). Linear logic for meaning assembly. In S. Manandha, P. G. Lops, & W. Nutt (Eds.), *Proceedings of Computational Logic for Natural Language Processing*. Edinburgh: Scottish Academic Press.

Dalrymple, M., Lamping, J., Pereira, F. C., & Saraswat, V. (1997b). Quantifiers, anaphora, and intensionality. *Journal of Logic, Language and Information, 6*(3), 219–273. Reprinted in M. Dalrymple (Ed.), *Semantics and syntax in Lexical Functional Grammar: The resource logic approach*. Cambridge, MA: MIT Press.

Dalrymple, M., Lamping, J., Pereira, F. C., & Saraswat, V. (1999). Quantification, anaphora, and intensionality. In M. Dalrymple (Ed.), *Semantics and syntax in Lexical Functional Grammar: The resource logic approach.* Cambridge, MA: MIT Press.

Dalrymple, M., Lamping, J., & Saraswat, V. (1993). LFG semantics via constraints. In *Proceedings of the sixth meeting of the european ACL* (pp. 97–105). University of Utrecht.

Frank, A., & van Genabith, J. (2001). LL-based semantics construction for LTAG- and what it teaches us about the relation between LFG and LTAG. In *Proceedings of the LFG01 Conference*. Stanford: CSLI Publications.

Fry, J. (1999). Proof nets and negative polarity licensing. In M. Dalrymple (Ed.), *Semantics and syntax in Lexical Functional Grammar: The resource logic approach*. Cambridge, MA: MIT Press.

Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science, 50*, 1–102.

Girard, J.-Y. (1989). *Proofs and types*. Cambridge: Cambridge University Press.

Howard, W. A. (1980). The formulae-as-types notion of construction. In J. Hindley, & J. Selden (Eds.), *To H.B. Curry: Essays on combinatory logic, lambda calculus and formalism*. New York: Academic Press. Conceived in 1969. Sometimes cited as Howard (1969).

Janssen, T. M. V. (1997). Compositionality. In J. van Benthem, & A. ter Meulen (Eds.), *Handbook of logic and language*. (pp. 417–473). Amsterdam: Elsevier.

Kaplan, R. M., & Bresnan, J. (1982). Lexical functional grammar: a formal system for grammatical representation. In J. Bresnan (Ed.), *The mental representation of grammar relations*. (pp. 173–281) Cambridge, MA: MIT Press.

Kokkonidis, M. (2005). Why glue your donkey to an f-structure when you can constrain and bind it instead? In M. Butt, & T. H. King (Eds.), *Proceedings of the LFG05 Conference*. Stanford: CSLI Publications.

Montague, R. (1974). *Formal philosophy. Selected papers of Richard Montague*. New Haven, Connecticut: Yale University Press. Edited and with an introduction by Richard H. Thomason.
Moorgat, M. (1997). Categorial type logics. In J. van Benthem, & A. ter Meulen (Eds.), *Handbook of logic and language*. Amsterdam: Elsevier.
Morrill, G. V. (1994). *Type logical grammar: Categorial logic of signs*. Dordrecht: Kluwer.
Potts, C. (2003). *The logic of conventional implicatures.* Ph.D. dissertation, University of California, Santa Cruz.
van Genabith, J., & Crouch, R. (1997). How to glue a donkey to an f-structure or porting a dynamic meaning representation language into LFG's linear logic based glue language semantics. In *Proceedings of the International Workshop for Computational Semantics* (pp. 52–65). Tilburg.